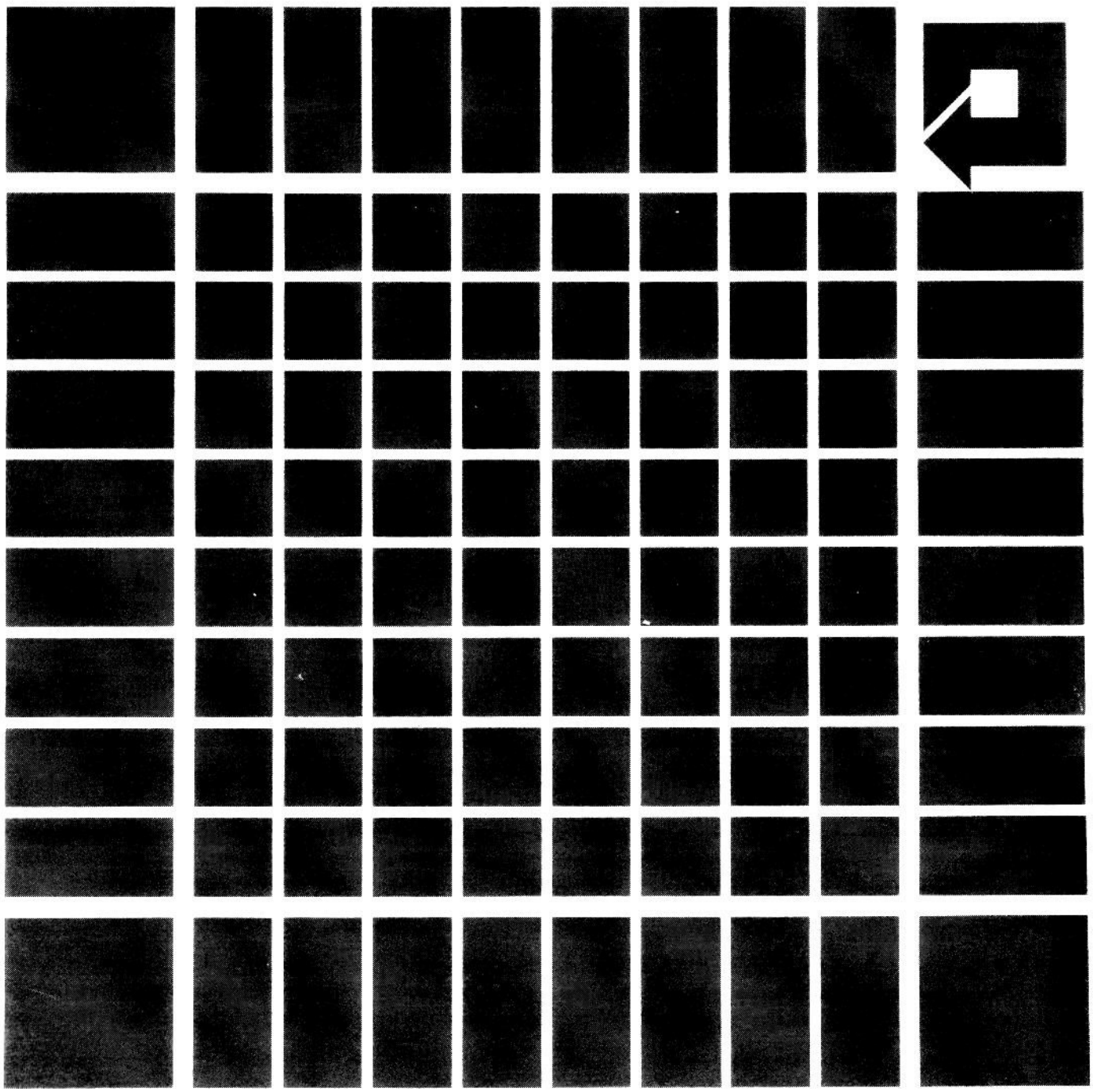
HEWLETT-PACKARD

HP-IL Application Note

Firmware Design for HP-IL Devices



Introduction

One of the problems you may encounter in executing an HP-IL interface design is the difficulty of translating the state diagrams, which are the official definition of the HP-IL protocol, into correct code for the microprocessor you are using. While the state diagrams are a very complete and concise way of specifying the protocol, this type of representation is not generally familiar to firmware designers. Furthermore, the HP-IL integrated circuit is fairly complex itself, and determining what parts of the protocol are handled automatically and what parts need to be taken care of in firmware is not always obvious.

The purpose of this application note is to assist the firmware designer by providing a detailed template for the HP-IL routine in the form of pseudo-code which is easy to understand and can be converted to microprocessor-specific code with a minimum of effort. If the template is accurately reproduced in the firmware, a correct implementation of HP-IL is virtually assured. In addition, the performance of the interface should also be relatively good since the template makes maximum use of the automatic features of the HP-IL integrated circuit.

To derive the full benefit of this application note, it is recommended that you have a basic familiarity with HP-IL protocol and the operation of the HP-IL integrated circuit as described in "The HP-IL Interface Specification" (82166-90017) and "The HP-IL Integrated Circuit User's Manual" (82166-90016). The small book, "The HP-IL System: An Introductory Guide to the Hewlett-Packard Interface Loop" (92233A), is a good place to start if you are not familiar with HP-IL. Clearly, you will also need a detailed understanding of your device and your particular microprocessor.

Characteristics of The HP-IL Device

The HP-IL routine in your device will depend heavily on the features you want the device to have from the point of view of the interface. A device which implements all possible features of HP-IL would have rather complex firmware and many of those features would be used only rarely or not at all. The set of protocol features in the template described in this application note was selected to handle the needs of the large majority of devices while maintaining clarity and compactness of the resulting code. A few hundred bytes of microprocessor machine code are all that is required to implement the HP-IL routine template as described in this application note.

Controller capability is not implemented by the template described here. Most designers are interested in connecting a device to an already existing controller via HP-IL, and consequently only need the talker and listener functions handled by this routine. The controller functions would substantially complicate the firmware as well, and obscure the important interactions between the HP-IL chip and the microprocessor.

All of the features in the talker function are supported, including data, status, device (model number) ID, and accessory (capability) ID. Other capabilities supported are basic service request, basic remote/local, basic auto address, power down, parallel poll, device clear and trigger, and device-dependent commands. The standard designation of this device as described in appendix A of "The HP-IL Interface Specification" would be as follows: **C0**; **T1,2,3,4**; **L1**; **SR1**; **RL1**; **AA1**; **PD1**; **PP1**; **DC2**; **DT1**; **DD1**.

Conceptually, the device hardware consists of three parts: the device-specific parts, the microprocessor, and the HP-IL chip with its associated components. The microprocessor and its ROM contain the control program for both the device and the HP-IL interface. In general, your device routine will treat the HP-IL routine as a "black box", communicating with it only through a few carefully-defined flags and variables. The internal design of the HP-IL routine is presented here in great detail, of course, so that you can convert it to correct code for your particular microprocessor without having to do that design yourself.

Overview of The Firmware

At the highest level, the firmware program consists of little more than a simple loop as shown in Listing 1. This is a polled approach; interrupts are not used in the microprocessor. Conversion to an interrupt-driven scheme will be discussed briefly at the end of this note. When the power is turned on, there are two initialization routines that are executed: one for the device-specific functions and one for the HP-IL routine. Then the program simply calls the device routine and the HP-IL routine repeatedly, checking the power-on flag each time around the loop. When the power-on signal becomes false, the program exits the loop and executes two shut-down routines, one for the device-specific functions and one for the HP-IL routine. Many devices will not need to sense the state of a power-on flag or shut down in the orderly manner indicated here. These devices can simply eliminate the loop exit test and the two power-off routines.

The device routine takes care of all the functions of the device except for I/O via HP-IL. The HP-IL routine handles all interaction with the HP-IL integrated circuit (except power-on and power-off). The interaction and communication between these two routines is discussed in detail in the following section.

The subroutine structure shown here is not really necessary, provided the program flow is not modified. For example, there might actually be only one power-on routine, coded in-line, which performs the functions of both power-on routines shown here. This hierarchy and structure make the pseudo-code easier to understand.

Listing 1: Top-level Firmware Structure

```
CALL device.power-on.routine
CALL hp-il.power-on.routine
DO
  { CALL device.routine
  { CALL hp-il.routine
  UNTIL power-on.flag = 0
CALL hp-il.power-off.routine
CALL device.power-off.routine
END
```

The coding conventions used in this note will generally be similar to PASCAL without worrying about details. Notational peculiarities will be clarified in the text. High-level language constructs which are not easily converted to simple machine code are avoided.

Communication With The HP-IL Routine

This section describes in detail the flags, variables, buffers, and constants that provide the interface between the device routine and the HP-IL routine. In addition, certain capabilities are provided by small device-specific subroutines which are conceptually part of the device routine but are called by the HP-IL routine. Each of these elements will be discussed in the following text and table 1. The communication provided is very complete within the constraints of the device characteristics mentioned previously. Of course, not all of these elements will be needed by all devices.

There are three different flags which indicate critical HP-IL interface states to the device routine: the listener flag, the talker flag, and the remote flag. These are set and cleared only by the HP-IL routine; they must never be modified by the device routine. The listener and remote flags can only take on values of one or zero, representing active or idle. The talker flag can take on several values since there are multiple states which must be represented. Zero represents the talker idle state, one represents the talker addressed state, two is talker active (data), and three is any one of serial poll (status), device ID, or accessory ID active states. The device routine may need to read these flags for various reasons. For example, the remote flag might be used to change the interpretation of the input from data to commands, or perhaps to disable the effects of manual controls on the device. Note that the remote flag is equivalent to the remote state (**REMS**) in the HP-IL remote/local interface function, not the remote active state (**RACS**).

There are two flags which allow the device routine to signal a need for attention to the HP-IL controller. These flags are set only by the device routine. The service request flag can be set to one to indicate a need for service. The HP-IL routine reads the state of this flag so that it can set the proper bit in certain frames on the loop. The service request flag may be reset to its normal value of zero either by the device routine or by the HP-IL routine. The parallel poll flag operates similarly except that HP-IL protocol does not allow it to be reset by the HP-IL routine. When a device condition occurs which requires a service request or parallel poll response, the device routine sets the appropriate flag to one. When the condition is satisfied, the device routine returns the flag to a zero value. A device which does not implement one or both of these functions would simply not use the corresponding flag.

The input buffer is an array of bytes which temporarily holds the data received from HP-IL until the device routine can process it. Likewise, the output buffer holds data that is in the process of being sent through the HP-IL routine to the loop. Both of the buffers operate in a first-in-first-out (FIFO) fashion. Since there are a number of ways to implement a FIFO buffer in firmware, the details of the implementation will not be specified here. You will need to determine how best to accomplish this for your particular microprocessor. The pseudo-code in this application note will simply indicate the actions of reading or writing these buffers, as well as checking whether or not they are empty or full. The size of the buffers will not be specified, since this will also be very dependent on the device requirements as well as available memory.

There are three other byte arrays which provide the HP-IL routine with the status, device ID, and accessory ID of the device. The two ID arrays are constants; they are set when the firmware is compiled and not modified by either the device routine or the HP-IL routine. The status byte array will contain either standard HP-IL status codes or device-dependent status bits, or both, and will need to be set to the correct values by the device routine. The HP-IL routine will not modify the status values but will simply send them on the loop at the proper time.

Table 1: Links to the HP-IL Routine

Mnemonic	Description
listener.flag device routine: **R** HP-IL routine: **RW**	This flag indicates whether or not the device may receive data from the loop. The device routine may read this flag, but must not modify it. A value of zero indicates the idle state (**LIDS**); a value of one indicates the active state (**LACS**). While in the active state, the HP-IL routine places received data bytes in the input buffer.
talker.flag device routine: **R** HP-IL routine: **RW**	This flag indicates whether or not the device may send data to the loop. The device routine may read this flag, but must not modify it. A value of zero indicates the idle state (**TIDS**), one is the addressed state (**TADS**), two is the active state (**TACS**), and three is any one of the serial poll (**SPAS**), device ID (**DIAS**), or accessory ID (**AIAS**) states. While in the active state, the HP-IL routine will remove bytes placed in the output buffer by the device routine and transmit them on the loop. The other states are handled automatically without intervention from the device routine.
remote.enable.flag device routine: **N** HP-IL routine: **RW**	This flag is used internally in the HP-IL routine and will not be needed by the device routine. A value of one indicates that the **REN** message has been received. A value of zero indicates the **NRE** message has been received. When this flag is one, the remote flag can change state; when this flag is zero, the remote flag is always zero.
remote.flag device routine: **R** HP-IL routine: **RW**	This flag indicates the remote control or local control state of the device. A value of zero indicates the local state (**LOCS**) and a value of one indicates the remote state (**REMS**). The device routine may read this flag, but must not modify it. Interpretation of this flag is device-dependent. The **GTL** command will cause this flag to be zero, as will a zero remote enable flag.
parallel.poll.flag device routine: **RW** HP-IL routine: **R**	The device routine may set or clear this flag depending on whether it needs to respond to a parallel poll request from the HP-IL controller. A value of zero causes a negative response and a value of one causes a positive response. The HP-IL routine reads this flag, but does not modify it.
service.request.flag device routine: **RW** HP-IL routine: **RW**	The device routine may set or clear this flag depending on whether it needs to send a service request to the HP-IL controller. A value of zero causes no request and a value of one causes a request to be sent. The device routine should set this flag to zero when the reason for requesting service is satisfied. The HP-IL routine will set it to zero when a serial poll (status) request is received, as the interface function state diagram specifies.
input.buffer device routine: **R** HP-IL routine: **W**	This is an array of bytes which operates in a first-in-first-out (FIFO) manner. Data from the loop for this device is placed here by the HP-IL routine. The device routine will read this data and use it as appropriate.
data.out.buffer device routine: **W** HP-IL routine: **R**	This is an array of bytes which operates in a first-in-first-out (**FIFO**) manner. Data from the device to be sent on the loop is placed here by the device routine. The HP-IL routine removes this data and transmits it on the loop when in the talker active state.
status-id.out.buffer device routine: **N** HP-IL routine: **RW**	This is an array of bytes which operates in a first-in-first-out (**FIFO**) manner. This buffer holds the status, device ID, and accessory ID bytes as they are being transmitted. The device routine does not use this buffer.

Table 1: Links to the HP-IL Routine *(continued)*

Mnemonic	Description
device.status device routine: **RW** HP-IL routine: **R**	The status of the device may consist of coded message bytes, individual flag bits within bytes, or both. Its length and content are device-dependent. The device routine will periodically update the status as appropriate, and the HP-IL routine will read and transmit this data to the loop when requested by the loop controller. Further information can be found in the HP-IL interface specification.
device.id device routine: **N** HP-IL routine: **R**	This constant is set when the firmware is compiled since there is no need to ever modify it. It usually contains the device manufacturer code and model number in ASCII, followed by the carriage return and line feed codes. The HP-IL routine sends the device ID on the loop when requested by the loop controller.
accessory.id device routine: **N** HP-IL routine: **R**	This constant is set when the firmware is compiled since there is no need to ever modify it. It usually contains a one-byte code indicating the type of device. The HP-IL routine sends the accessory ID on the loop when requested by the loop controller. Further information can be found in the HP-IL interface specification.
default.address.byte device routine: **N** HP-IL routine: **R**	This constant defines the address at which the device will respond after an **AAU** message is received. The device routine will not use this value. At power-up, the address register is loaded with the value 31 (an illegal address) and will not respond. When **AAU** is received, the device may choose to respond to some default address rather than remain unconfigured.
ddl.routine	This subroutine executes the device-dependent listener commands that the device implements. It is called by the HP-IL routine when one of this class of commands is received and the device is in the listener active state. The subroutine may use the input.byte variable to determine which one of the 32 commands is to be executed.
ddt.routine	This subroutine executes the device-dependent talker commands that the device implements. It is called by the HP-IL routine when one of this class of commands is received and the device is in the talker addressed state. The subroutine may use the input.byte variable to determine which one of the 32 commands is to be executed.
input.byte device routine: **R** HP-IL routine: **RW**	This variable holds the data bits of the most recently received frame. It has a range of values from zero to 255. It can be read by the device routine, but must not be modified by it.
interrupt.byte device routine: **N** HP-IL routine: **RW**	This variable holds the interrupt bits from the integrated circuit as well as the control bits from the received frame. It is only used by the HP-IL routine for decoding.
output.byte device routine: **N** HP-IL routine: **RW**	This variable holds the data bits of the most recently transmitted byte. It is only used by the HP-IL routine to do manual error-checking in those few instances that this is needed.
device.clear.routine	This subroutine executes the device clear or selected device clear command. It is called by the HP-IL routine when one of these two commands is received from the loop.
device.trigger.routine	This subroutine executes the device trigger command. It is called by the HP-IL routine when this command is received from the loop.

Certain commands are executed differently depending on the device. Therefore the device routine must supply a subroutine for each of these commands which can be called by the HP-IL routine when that particular command is received. Device clear and device trigger are two of these. In addition, HP-IL provides for up to 32 device-dependent listener commands (**DDL**) and up to 32 device-dependent talker commands (**DDT**). The number of the particular command will be held in an input byte which can be read by the device-dependent command subroutine, but must not be modified. The subroutines do not need to check the listener or talker flags since this is done automatically by the HP-IL routine before the subroutine is called. The **DDL** and **DDT** subroutines will decode the input byte and perform the functions defined by that command or simply return if that particular command is not implemented by the device.

The communication interface between the device routine and the HP-IL routine is summarized in Table 1. **R** indicates read-only, **W** indicates write-only, **RW** indicates read-write, and **N** indicates not used. For example, "device routine: **R**" means the device routine may read the variable, but must not modify it. "HP-IL routine: **RW**" means the HP-IL routine may either read or write the variable as needed. Table 1 is a complete listing of all variables used by the HP-IL routine as well as all external calls made by it.

The HP-IL Power-on And Power-off Routines

The first thing you need to take care of after power is applied is the initialization of the device and the HP-IL chip. You will need to be sure to initialize the device status and you probably will need to do some other things as well to get the device ready to operate. This section will describe what is needed as far as the HP-IL chip is concerned.

Usually the /RESET line on the HP-IL chip is tied to the line of the same name on the microprocessor. When power is applied, this line is held low for a short time and then goes high. This is fairly important to prevent anomalous operation of the chip before the microprocessor can reset it in the program. If this signal has a slow rise time, however, you may experience some problems. The processor may be ready to go sooner than the HP-IL chip if the processor's threshold is somewhat lower on this line. Buffering the /RESET line should make the rise time fast enough that both chips will be ready when the power-on routine is executed.

The first thing the power-on routine does is set the master clear bit in the HP-IL chip just for redundancy. The next thing to do is turn on the chip oscillator. It is important to give the oscillator time to stabilize before sending or receiving any HP-IL frames, so a wait is placed in the routine at this point. Since master clear does not disable the parallel poll response or clear the address register, it is important to do this also before the chip begins normal operation. After the state flags and the buffers are cleared, the master clear bit is reset and the chip is ready to go.

Listing 2 contains the HP-IL power-on routine. The symbol := is used for the assignment operator. The registers of the HP-IL chip are indicated by R0, R1, R2, etc. Binary numbers are shown in brackets; all other numbers are decimal. For example, the statement, "**R0 := [00000001]**" indicates a write to register zero with the binary value 00000001 (bit zero is set to one). The statement, "**input.byte := R2**" means to read register two of the HP-IL chip and place the value in the input.byte variable.

Listing 2: The HP-IL Power-on Routine

```
R0 := [00000001]            set master clear
R7 := [00000000]            turn on oscillator
WAIT 1                      wait 1 msec for oscillator
R3 := [00000000]            disable parallel poll
R4 := [00011111]            address unconfigured
listener.flag := 0          clear state flags
talker.flag := 0
remote.flag := 0
remote.enable.flag := 0
CLEAR input.buffer          clear buffers
CLEAR data.out.buffer
CLEAR status-id.out.buffer
R0 := [00000000]            reset master clear
EXIT
```

The power-off routine, if needed, is quite simple. Master clear is set and the oscillator is turned off. Listing 3 shows this routine. If the /RESET line goes low, these same operations take place automatically. The device may also require some operations at power-off, but you will have to write these as part of the device power-off routine.

Listing 3: The HP-IL Power-off Routine

```
R0 := [00000001]            set master clear
R7 := [00000001]            turn off oscillator
EXIT
```

Overview of The HP-IL Routine

Now that the preliminaries have been taken care of, it is time to look into the routine that actually handles the HP-IL chip and the messages on the loop. Because the routine is larger than the others, it has been broken down into two levels. This section will discuss the top level, and the succeeding sections will cover the various modules on the lower level.

When the HP-IL routine is first entered, the parallel poll and service request flags are checked and the chip registers are set accordingly. Then the interrupt register is read to determine if any action is required. This level simply executes one of four lower level routines to service each of the possible interrupt causes.

Please note that the order of execution of the bit tests is important. Indeed, this is generally true of the programs in this application note. You should duplicate them as closely as possible without any change in the statement order. In order to set or clear individual bits in registers, the AND and OR functions are used. These are the usual bit-by-bit logical operations that are used by microprocessors. Also, the symbol <> is the "not equals" relational operator. The left brace { is used to set off a block in the program that is executed or skipped as a unit.

It is very important not to set the **SLRDY** bit in register zero unless that is really what is intended. When this register is read, the bit in the **SLRDY** position is the **RFCR** bit. To prevent any possibility of a problem, the **SLRDY** position is always explicitly set to zero when register zero is written, except when the intent is specifically to set **SLRDY**.

Listing 4: The Top Level of the HP-IL Routine

```
IF parallel.poll.flag = 1
  THEN R3 := R3 OR [00100000]                          set PPST
  ELSE R3 := R3 AND [11011111]                         clear PPST
IF service.request.flag = 1
  THEN R0 := (R0 AND [11111011]) OR [00001000]         set SSRQ, but not SLRDY
  ELSE R0 := (R0 AND [11110011])                       clear SSRQ, SLRDY
interrupt.byte := R1
IF (interrupt.byte AND [00010000]) <> 0
  THEN
  { CALL ifcr.routine
  { EXIT
IF (interrupt.byte AND [00000100]) <> 0
  THEN
  { CALL frav.routine
  { EXIT
IF (interrupt.byte AND [00000010]) <> 0
  THEN
  { CALL frns.routine
  { EXIT
IF (interrupt.byte AND [00000001]) <> 0
  THEN
  { CALL orav.routine
  { EXIT
EXIT
```

The Interface Clear Routine

Since the interface clear command only affects the state of the HP-IL routine, there is no need for the device routine to even be aware of this bit. The routine to handle the **IFCR** bit is given in Listing 5.

Listing 5: The IFCR Routine

```
listener.flag := 0
talker.flag := 0
input.byte := R2                                       clears FRAV, FRNS
R0 := (R0 AND [00001000]) OR [00000110]                clear TA, LA; set SLRDY, CLIFCR; do not
                                                       change SSRQ

EXIT
```

The Frame Available Routine

Listing 6 contains the frame available (**FRAV**) routine. This block of code takes care of data frames while the device is an active listener, ready frames, and commands. Rather than list the binary code for each of the HP-IL messages, the three-letter mnemonic will be used for clarity.

Listing 6: The FRAV Routine

```
IF (interrupt.byte AND [10000000]) = 0                    data frame?
  THEN                                                    this section handles data frames
  { IF input.buffer FULL THEN EXIT
  { input.byte := R2
  { WRITEBYTE FROM input.byte TO input.buffer
  { R2 := input.byte                                      retransmit data frame
  { EXIT
input.byte := R2
IF (interrupt.byte AND [00100000]) = 0                    command frame?
  THEN                                                    this section handles commands
  { IF (input.byte AND [11100000]) = LAD0
  {   THEN
  {   { IF input.byte = UNL
  {   {   THEN
  {   {   { R0 := R0 AND [11101011]                       clear LA, SLRDY
  {   {   { listener.flag := 0
  {   {   ELSE
  {   {   { IF (input.byte AND [00011111]) = (R4 AND [00011111])
  {   {   {   THEN                                        address match; set LA; clear TA, SLRDY
  {   {   {   { R0 := (R0 AND [11011011]) OR [00010000]
  {   {   {   { talker.flag := 0
  {   {   {   { listener.flag := 1
  {   {   {   { IF remote.enable.flag := 1 THEN remote.flag := 1
  { IF (input.byte AND [11100000]) = TAD0
  {   THEN
  {   { IF (input.byte = UNT) OR ((input.byte AND [00011111]) <> (R4 AND [00011111]))
  {   {   THEN                                            UNT or no match; clear TA, SLRDY
  {   {   { R0 := R0 AND [11011011]
  {   {   { talker.flag := 0
  {   {   ELSE                                            address match; set TA; clear LA, SLRDY
  {   {   { R0 := (R0 AND [11101011]) OR [00100000]
  {   {   { listener.flag := 0
  {   {   { talker.flag := 1
  { IF (input.byte = GET) AND (listener.flag = 1) THEN CALL device.trigger.routine
  { IF (input.byte = DCL) OR ((input.byte = SDC) AND (listener.flag = 1)) THEN CALL device.clear.routine
  { IF ((input.byte AND [11100000]) = DDL0) AND (listener.flag = 1) THEN CALL ddl.routine
  { IF ((input.byte AND [11100000]) = DDT0) AND (talker.flag = 1) THEN CALL ddt.routine
  { IF input.byte = AAU THEN R4 := default.address.byte
  { IF ((input.byte AND [11110000]) = PPE00) AND (listener.flag = 1)
  {                                                       set PPEN, PPPOL, P2-0; don't change PPST
  {   THEN R3 := (R3 AND [00100000]) OR input.byte OR [00010000]
  { IF (input.byte = PPU) OR ((input.byte = PPD) AND (listener.flag = 1))
  {   THEN R3 := R3 AND [11101111]                        clear PPST
  { IF input.byte = REN THEN remote.enable.flag := 1
  { IF input.byte = NRE
  {   THEN
  {   { remote.enable.flag := 0
  {   { remote.flag := 0
```

Listing 6: The FRAV Routine *(continued)*

```
    { IF (input.byte = GTL) AND (listener.flag = 1) THEN remote.flag := 0
    { IF input.byte = LPD
    {   THEN                                        wait for RFC; exit if anything else
    {   { DO IF (R1 AND [00010110]) <> 0 THEN EXIT
    {   {   UNTIL (R0 AND [00000100]) <> 0
    {   { R0 := [00000100]                          set SLRDY to send RFC
    {   { WAIT .1                                   wait .1 msec to send RFC
    {   { power-on.flag := 0
    {   { EXIT
    { R0 := R0 OR [00000100]                        set SLRDY bit for RFC
    { EXIT
  IF talker.flag = 1                                this section handles ready frames
    THEN
    { IF input.byte = SDA
    {   THEN
    {   { talker.flag := 2
    {   { EXIT
    { IF input.byte = SST
    {   THEN
    {   { CLEAR status-id.out.buffer
    {   { WRITEALL FROM device.status TO status-id.out.buffer
    {   { service.request.flag := 0
    {   { R0 := R0 AND [11110011]
    {   { talker.flag := 3
    {   { EXIT
    { IF input.byte = SDI
    {   THEN
    {   { CLEAR status-id.out.buffer
    {   { WRITEALL FROM device.id TO status-id.out.buffer
    {   { talker.flag := 3
    {   { EXIT
    { IF input.byte = SAI
    {   THEN
    {   { CLEAR status-id.out.buffer
    {   { WRITEALL FROM accessory.id TO status-id.out.buffer
    {   { talker.flag := 3
    {   { EXIT
  IF ((input.byte AND [11100000]) = AAD0) AND ((R4 AND [11100000]) = 0) AND (input.byte <> IAA)
    THEN                                            if not configured, set address and configured
    {                                               flag (high order bit in AAD frame)
    { R4 := input.byte
    { input.byte := input.byte + 1
  R2 := input.byte                                  retransmit ready frame
  EXIT
```

In order to be in the **FRAV** routine and receive a data frame, the device must already be an active listener. If it was not, the data frame would be automatically retransmitted and would not even be noticed by the chip. If the input buffer is full when another data frame is received, the HP-IL routine merely exits, holding the frame until the device routine removes one or more bytes from the input buffer. Then the data frame is placed in the input buffer. If this happens, the HP-IL handshake is temporarily stopped.

The command and ready frame processing is fairly straightforward. Unrecognized frames are simply retransmitted. The decoding of each command and ready frame is independent. If your device does not implement a particular frame, you can usually just leave out the statements that refer to that message without affecting other parts of the code.

Please note that the device routine can use registers five and six on the HP-IL chip as general purpose storage. The three high-order bits of register four must not be used, however, since they are used by the HP-IL routine to indicate whether or not the **AAD** message has been received.

The Frame Received Not As Sent Routine

In those situations where this device is the source of frames, the chip automatically error-checks the returning frame to see that it is the same as the frame which was sent. If it is not, the **FRNS** interrupt bit is set so that the microprocessor can take appropriate action. In most cases, the response is to simply terminate the transmission with the **ETE** message and let the loop controller handle the problem. That is how the HP-IL routine responds here.

The **FRNS** routine is shown in Listing 7. The routine sends either the **ETO** or the **ETE** message to signal the end of the transmission and sets the talker flag back to the addressed state. The correct control bits for a ready frame are written to register one before the frame is sent.

There is one case where error-checking must be performed by the firmware. Notice that there is no provision to process the **NRD** message in the **FRAV** routine. It is simply retransmitted. When the data frame which was held by the device sending the **NRD** returns, the **FRNS** bit will be set. The chip cannot do the checking since the transmit register has been modified by the **NRD**, and so this is done by the **FRNS** routine. In this situation, an error has not actually occurred and the **ETO** message will be sent. Also notice that the same thing would happen if the **NRD** were replaced by some other undefined ready frame. While this is not strictly according to protocol, the situation will never happen if the other devices are obeying protocol.

Listing 7: The FRNS Routine

```
input.byte := R2
R1 := [10100000]              set RDY control bits
IF input.byte = output.byte
  THEN R2 := ETO
  ELSE R2 := ETE
talker.flag := 1
EXIT
```

The Output Register Available Routine

When the device is the active talker and a data byte which was transmitted returns and error-checks correctly, the **ORAV** bit is set. This indicates that the HP-IL routine should send the next byte. The **ORAV** routine is shown in Listing 8. If the HP-IL chip is not the talker, the **ORAV** bit will always be set. The first line of this routine takes care of that situation.

Listing 8: The ORAV Routine

```
IF talker.flag < 2 THEN EXIT
R1 := [00000000]                                    set DAB control bits
IF talker.flag = 2
  THEN                                              data transmission
  { IF data.out.buffer EMPTY
  {   THEN
  {   { R1 := [10100000]                            set RDY control bits
  {   { output.byte := ETO
  {   { talker.flag := 1
  {   ELSE READBYTE FROM data.out.buffer TO output.byte
  ELSE                                              status, device ID, or accessory ID
  { IF status-id.out.buffer EMPTY
  {   THEN
  {   { R1 := [10100000]                            set RDY control bits
  {   { output.byte := ETO
  {   { talker.flag := 1
  {   ELSE READBYTE FROM status-id.out.buffer TO output.byte
R2 := output.byte                                   send the frame
EXIT
```

A Word About Interrupts

While the simple polled approach will be adequate for most applications, some will have a need to be interrupt-driven. This conversion of the routine is relatively easy, but there are a several details to watch out for.

The logic in the chip causes the ORAV bit to stay set all the time whenever the **TA** and **CA** bits are clear. This means that unless the interrupt is masked with the interrupt-enable bits in register one, the routine will be locked in a tight loop continually interrupting whenever the device is not the talker. In the routine described in this note, the enable bits are all cleared since the interrupt line is not used. In an interrupt-driven routine, these bits would be set so that whenever any one of the interrupt bits was set, an interrupt would be generated by having the HP-IL chip drive the /**IRQ** line low. Whenever the **TA** bit is set, all the enable bits are set to one. When the **TA** bit is not set, however, the **ORAV** enable bit must be cleared so continual interrupting does not occur.

A similar situation can arise if the device is the listener and the input buffer is full. Here the routine purposely leaves the frame in register two without reading it, which leaves the **FRAV** bit set. In this case, continual interrupts will occur unless this bit is masked until the device removes some of the bytes from the input buffer and allows the HP-IL routine to continue normal operation. Since masking the **FRAV** bit will not allow the interrupt to occur again, the device routine will have to explicitly call the HP-IL routine when it has made room in the buffer.

Portable Computer Division
1000 N.E. Circle Blvd., Corvallis, OR 97330, U.S.A.

European Headquarters
150, Route du Nant-D'Avril
P.O. Box, CH-1217 Meyrin 2
Geneva-Switzerland

HP-United Kingdom
(Pinewood)
GB-Nine Mile Ride, Wokingham
Berkshire RG11 3LL

82166-90024 English

Printed in U.S.A. 3/84